

Priority Interrupts of Duty Cycled Communications in Wireless Sensor Networks

Tony O'Donovan¹, Jonathan Benson¹, Utz Roedig², Cormac J. Sreenan¹

¹Mobile and Internet Systems Laboratory, Dept. of Computer Science, University College Cork, Ireland.

²InfoLab21, Lancaster University, UK.

Abstract— FrameComm is a contention based, Duty Cycled, MAC protocol that ensures a message will be transmitted during the receiver's listen phase by sending a packet, followed by a short gap, repeatedly for a precalculated number of times or until an acknowledgment is received. While introducing duty cycled communications can yield large power savings it does so at the cost of increased delay and decreased throughput. Many WSNs may incorporate several distinct message types of varying priority. A node with a high priority message to send may find the channel to be busy with a lesser priority message from another node and must therefore 'back-off' leading to further delays. In a multi-hop environment, these delays are compounded and may become unacceptably large. This paper proposes adding a high priority interrupt message to FrameComm that allows a node with important data to send to interrupt another node's lesser priority transmission giving immediate access to the channel. The priority interrupt mechanism is evaluated using an implementation in TinyOS 2 on a small laboratory testbed.

I. INTRODUCTION

Communication in Wireless Sensor Networks (WSNs) consumes a great deal of energy and has been the focus of much research with many energy efficient communication protocols being proposed. Many WSN platforms use the capabilities of modern low power radio transceivers, such as ChipCon's CC2420, to achieve substantial power savings. These radios generally have four modes of operation, listen, receive, transmit and sleep. Energy efficiency is maximized by spending as much time as possible in sleep mode which consumes only μA compared to mA when using the other modes. However, since communication between two nodes cannot occur if either node's radio is in sleep mode, a mechanism is required to ensure the sender's transmit and the receiver's listen operations coincide. This coordination of send and listen activities is known as transmitter-receiver rendezvous.

If the listen/sleep cycle of the receiver is fixed and known then a transmission can be simply extended such that an overlap between the transmission and the listen phase is guaranteed to occur. Traditionally, this was achieved using a long preamble immediately followed by the packet payload. A receiver hearing the preamble would keep its radio in a listen state until the packet was received. This duty cycle concept was implemented in B-MAC [1]. A number of modern radios such as the CC2420 provide additional features such as automatic packet header processing and CRC computation and are normally used to send complete packets rather than individual bits. With these transceivers it is possible for a sender to transmit a series of identical short messages,

which we call 'Framelets', in such a way that the receiver is guaranteed to catch at least one. Despite the overhead for the sender the scheme is very energy efficient, requires no additional hardware and no active coordination among nodes. This duty cycle concept was implemented in FrameComm [2] and is also used in the current TinyOS LPL low power listening module.

A WSN has to transport a number of different message types. Control information is exchanged and data readings from a variety of sensors are collected. Naturally, all of these messages may be of different significance and it would be useful to prioritize their transport within the sensor network accordingly. A scheduling mechanism can be employed to order messages according to their priority before transmission. However, even if a high priority message is scheduled immediately, the channel may be blocked by low priority traffic from another node currently transmitting. In this case the node is forced to back-off and wait until the channel becomes available. In a duty-cycled communication system the channel might be blocked for a significant duration. Additionally, the message may have to travel several hops in the network and these waiting times might occur at a number of hops resulting in large end-to-end delays for high priority messages. To solve this problem, we propose a method that allows the interruption of ongoing transmissions to gain access to the channel promptly if a high priority message needs to be sent.

We propose the use of inter-framelet interrupts to implement priority channel allocation. An interrupt can be sent by a node with a higher priority message if a lower priority message from a neighboring node is currently transmitting its framelet trail. The interrupt is sent during the gap between framelets and is acknowledged by the node currently transmitting. This node then enters a back-off state leaving the channel clear for transmission of the high priority message. This simple but effective system can be used to prioritize varying traffic types.

This paper presents the basic concept of priority interrupts for framelet based communication systems. An implementation of the concept using our previously developed FrameComm [2] medium access control protocol is presented. Some practical application areas where the scheme would be of benefit are also described. Finally, the effectiveness of the interrupt mechanism is evaluated within a small WSN lab deployment.

II. RELATED WORK

Several other Duty Cycle based MAC protocols use Framelet trails to ensure the listen and send activities overlap allowing communication to occur. The current default energy saving protocol in TinyOS[3] is based on the Low Power Listening component of [1], but uses message retransmission instead of a long preamble in order to accommodate packet based radios like that of [4]. X-Mac [5] also uses framelets to establish rendezvous between sender and receiver but only retransmits a message header, the payload is sent only after one of the replicas has been acknowledged and the sender knows that the destination is listening. Koala [6] is another Duty Cycled scheme, here the sender listens to the channel for a ‘probe’ message that the receiver sends each time it wakes up, when the receiver’s probe message is acknowledged it stays awake for the incoming transmission. All of these protocols are primarily focused on energy-efficiency and do not take message priority into account.

The requirements for low-latency, hard real-time applications and the potential for 802.15.4 based radios to satisfy them is examined in [7]. The problem of scheduling access to the wireless medium is the focus of most of the related work, ensuring messages with high priority can gain access to the channel first as in [8]. PR-MAC[9] is among many protocols that use Arbitration Inter-Frame Space (AIFS) from IEEE 802.11e[10] to prioritize channel access. Adaptive Distributed Fair Scheduling (ADFS) described in [11] incorporates a weighted queue to schedule messages based on priority, however the protocol requires synchronization of nodes to operate. Few works address the problem of interrupting an already ongoing transmission on the shared medium. In particular, to the best of our knowledge, no work addresses the problem of interrupting an ongoing transmission in the context of duty cycled wireless sensor networks. RTQS[12] presents an approach to conflict free transmission scheduling and provides prioritised communications, however it requires static priorities. CoBRA [13] is a framework that supports multiple classes of traffic in sensor networks by enforcing rate control using distributed cluster based mechanisms.

TOMAC [14] is a TDMA based MAC protocol that uses a non-destructive bit-wise arbitration for message preambles. In a TDMA slot all nodes with data to send start simultaneously to transmit their unique bit sequence as preamble. The transmission of a 0 is dominant and as a result the node with the lowest number transmitted as preamble will win and continue to transmit the data block. Channel access can be prioritized according to the bit numbers used as preamble. An ongoing transmission is not interrupted, instead, all transmitters start simultaneously and the transmitter with the highest priority will continue to the end while others back-off.

Brown et al. [15] describe a sensor node that uses two radios. A low powered radio with small bandwidth is used as a wake-up radio, data transmission follows on the high powered radio with larger bandwidth. Their work describes the usage of the wake-up radio to schedule transmissions but it would

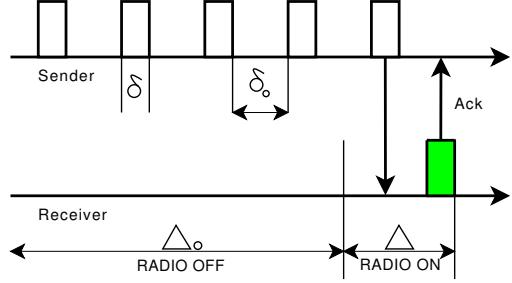


Fig. 1. Framelet-rendezvous.

easily be possible to use the same concept to interrupt ongoing low priority transmissions. However, such a solution requires the use of two radio devices on the sensor node.

III. FRAMELETS

In this Section, the Framelet rendezvous mechanism is outlined (see [16] for details). The relationships between the packet transmission time δ , the gap between transmissions δ_0 , the receiver listening time Δ and the number of required framelets n are defined. These relationships are crucial to ensure correct rendezvous between transmitter and receiver. The rendezvous process is depicted in Fig. 1.

A. Duty Cycle

The duty cycle period is represented as $P = \Delta + \Delta_0$, where Δ is the time the radio that remains active and Δ_0 is the time that the radio spends in sleep mode. The duty cycle ratio, or duty cycle D for short, is defined as:

$$D = \frac{\Delta}{P} = \frac{\Delta}{\Delta + \Delta_0} \quad (1)$$

B. Rendezvous using Framelets

Framelets are defined as small packets having a limited size. The trail of framelets is characterized by three parameters:

- Number of transmissions: n
- Time between framelets: δ_0
- Framelet transmission time: δ

In order to achieve correct transmitter-receiver rendezvous using framelets a number of relationships must be obeyed.

First, there is a relationship between the transmission time of a framelet, δ , and the gap in between framelet transmissions, δ_0 , to the minimum listen time at the receiver, Δ . This ensures that the listener is active for a sufficient time to guarantee overlap with at least one full framelet, assuming the channel is busy. Formally stated:

$$\Delta \geq 2 \cdot \delta + \delta_0 \quad (2)$$

With some transceivers δ may vary according to the length of a framelet. If δ or δ_0 are not constant then the worst case values can be assumed.

Second, there is a relationship between the minimum number of framelets, n , in a trail and the parameters of the duty

cycle, Δ and Δ_0 , and of course the parameters of the framelet trail itself, δ and δ_0 . This ensures that there are a sufficient number of framelets transmitted such that a correct overlap with a receiver's listening period, Δ , is guaranteed to occur. Formally stated:

$$n \geq \left\lceil \frac{P}{\delta + \delta_0} \right\rceil \quad (3)$$

If δ or δ_0 may vary it is difficult to calculate a priori the number of framelets needed. An alternative to this approach is to simply ensure that the transmission of the framelet trail exceeds the duration of P .

C. Acknowledgments

To save further energy on the transmitter side, an acknowledgment mechanism can be used. The radio switches to a listen state in the transmission gaps, δ_0 . A receiver acknowledges the reception of the first framelet encountered. After receipt of the acknowledgment the sender stops the framelet transmission. Thus, using acknowledgments, most transmissions will not require all n framelets. As a result, a transmission will occupy the channel for a shorter period of time. To enable acknowledgments, δ_0 must be long enough to cater for transceiver switching times and sending of the acknowledgment.

D. Fragmentation

One of the requirements for this framelet communication scheme is that there is a maximum limit on packet size, δ . Various factors such as the operating system and the specification of the node's radio transceiver can have an impact on the value chosen for δ . If the size of the data a node has to send exceeds δ , then it becomes necessary to split the data into several fragments that are smaller than the imposed limit and send each piece in a separate message. This can be done with a fragmentation layer.

The fragmentation layer adds a two byte fragmentation header to the message payload, consisting of a one byte length field and a one byte fragment number. The length field is used to denote the total number of fragments in the message and the fragment number indicates the position of the current fragment in the message. Transmission of a fragmented message begins with the message passing through the fragmentation layer where it is split into several pieces, each one being given a fragment number before being queued for sending.

The first fragment is sent like any other framelet, firstly listening to the channel to determine whether or not it is occupied before repeatedly sending the first fragment. The destination node will receive one of these framelets during the listen period of its duty cycle and send an acknowledgment. It then checks the fragmentation header of the framelet to determine if there are more fragments to come and extending its listen period if necessary.

Once the sender has received the acknowledgment for the first fragment it knows the destination node is listening and immediately sends the remaining fragments; since the

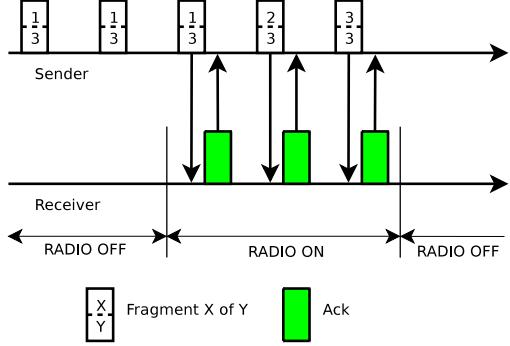


Fig. 2. Fragmentation.

destination is listening only a single framelet is required for each. The process can be seen in Fig. 2.

E. Summary

The duty cycle ratio, D (eq. 1), strongly influences the energy consumption pattern of a node; a small D indicates a low energy consumption. The worst-case message transfer delay between two nodes is defined by P . P can be reduced while keeping the required duty cycle D by minimizing the values of δ and δ_0 . Hence, to optimize message transfer delays, δ and δ_0 should be as small as possible. Depending on the capabilities of radio transceiver and the minimum packet size, lower bounds for δ and δ_0 exist.

IV. HIGH PRIORITY INTERRUPTS

An ongoing framelet transmission can be interrupted by any node in communication range by sending a short message similar to the acknowledgment described in Section III-C. This feature can be used by nodes that have to send high priority data and need to access the channel immediately.

A. Interrupt Concept

If a node has data to send it will attempt to access the channel. To do this it must first perform a carrier sense listen of duration Δ to ensure no other node is transmitting. If, during this listen phase, it receives a single framelet sent by a current transmitter, the header of this framelet is examined. If the priority is equal to or higher than the message in the transmit buffer then the usual back-off occurs. On the other hand, should the priority of the received framelet be lower than the message awaiting transmission then a priority interrupt will be performed. The node sends an interrupt packet to the current transmitter. The current transmitter receives this interrupt packet as it would be waiting for an acknowledgment from the destined receiver of the framelet transmission. If the current transmitter correctly receives this interrupt it will send an interrupt acknowledgment, cease transmissions and then enter a back-off phase. After this backoff the interrupted node will attempt to send its message again. Upon receipt of the interrupt acknowledgment the interrupter will immediately begin transmitting its own framelet trail. If an interrupt

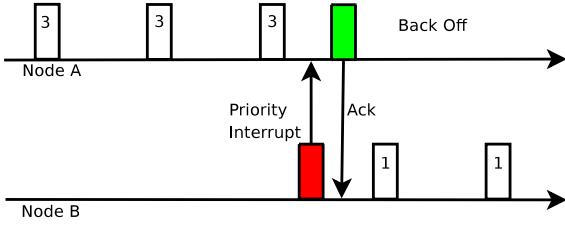


Fig. 3. High Priority Interrupts.

acknowledgment is not received the interrupter will enter a back-off phase.

The basic mechanism can be seen in Fig. 1. Node B has a high priority message to send and interrupts Node A who is transmitting lower priority messages.

B. Interrupt Acknowledgment

It is evident that this procedure could be performed without the use of an interrupt acknowledgment. However, the acknowledgment ensures that the interrupt mechanism works correctly and prevents potential collisions in the event of interrupt loss. In addition if multiple nodes attempt to interrupt simultaneously or nearly simultaneously then at most one will receive an acknowledgment and the rest will back-off. If all the interrupts collide then no acknowledgment will be sent and all the interrupting nodes will back-off.

C. Arbitration

This priority interrupt mechanism can be used with several levels of priority allowing multiple interruptions in a single duty cycle period. Any node that has a message to send can interrupt an ongoing lower priority framelet trail from a neighboring node. A node, Node B, that successfully interrupts another, Node A, can subsequently be interrupted by a third node, Node C, provided that Node C's message priority exceeds that of Node B. In effect this provides priority arbitration, ensuring the highest priority message gets access to the channel.

An example of this behavior can be seen in Fig. 4, Node A initially acquires the channel with a message of priority 3, Node A is interrupted by Node B whose message priority is 2. Node C then senses a level 3 event and attempts to report this with a message with priority 3 but finds the channel occupied by higher priority Node B so it backs off. Finally Node D, message priority 1, interrupts Node B. This example is a typical scenario and illustrates how the node with the highest priority message gets access to the channel without having to wait for the sender currently occupying the channel to deliver its message.

D. Summary

The process of interrupting another node takes place very quickly and the interrupting node can seamlessly take control of a busy channel to transmit its own high priority message. Multiple interrupts may take place among a group of nodes

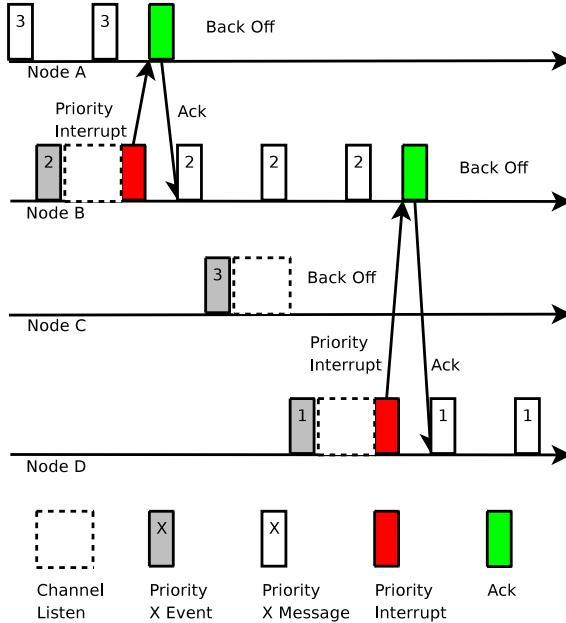


Fig. 4. Priority Arbitration.

(e.g. A interrupts B, which is then interrupted by C etc.). Thus, it is ensured that at a given time the highest priority message is very likely to gain access to the channel. For a given Duty Cycle Period P a node with a high priority message to send and only lower priority messages to contend with should always experience delays less than P and on average $\frac{P}{2}$. This observation can be expanded to multi-hop scenarios of h hops; here these values are $h \cdot P$ and $h \cdot \frac{P}{2}$ respectively.

V. IMPLEMENTATION

The previously described concepts were implemented in TinyOS 2.0.2 for Tmote Sky and Tmote Invent sensor nodes and denoted *FrameComm*. *FrameComm* was modified to include high priority interrupts; we refer to this protocol as *FrameCommHPI*.

A. FrameComm

FrameComm [2] is an implementation of a CSMA based MAC using the framelets scheme. It is similar to LPL (Low Power Listening), the default power saving protocol that is part of TinyOS 2.0.2, but it was developed in parallel and has a few key distinctions. In terms of similarities both concepts use framelet trails to ensure rendezvous and the listening cycle length is dictated by the duty cycle and the times δ and δ_0 .

LPL however does not implement the listening phase Δ in the same manner as *FrameComm*. LPL repeatedly polls the CCA to detect if the channel is busy for a fixed time of 11ms. If during this time a packet is detected the listening phase is extended to ensure that the packet is received correctly. *FrameComm* on the other hand performs a standard listen for 12ms, which should guarantee interception of any packets transmitting and performs a brief CCA at the end of the listen

period in case another framelet has begun transmitting. If so the listen period is extended. As a result of the longer listen period Δ FrameComm had a longer sleep period Δ_0 than LPL. A 2% duty cycle using FrameComm has a period $P = 600ms$ whereas LPL has a period $P = 550ms$. Therefore LPL should theoretically be able to handle more traffic than FrameComm at the same duty cycle.

Another major difference is the way in which the carrier sense is performed. Whereas FrameComm uses a full listen, as described above, LPL merely uses a brief CCA before beginning its transmissions. The use of this brief CCA is insufficient to ensure that the channel is in fact clear and a number of problems arise when LPL experiences contention for the channel (see Section VII).

Yet another difference between the two implementations is the back-off strategy. FrameComm makes use of a full listen and exploits the overheard information to influence its back-off strategy. LPL does not do this since it does not receive packets during its carrier sense phase. In addition the back-off strategy of LPL is unrelated to the duty cycle and results in numerous back-offs which are not sufficient to remove unnecessary contention for the channel.

B. FrameCommHPI

When a node has data to send it will pick up any message currently occupying the channel in its carrier sense phase. The priority of this message is ascertained by checking its type and comparing it to that of the message the node wishes to send. If the node finds that the current message in transit is of equal or higher priority it begins a back-off, allowing the current sender to complete the ongoing transmission. Otherwise, the node generates and sends an interrupt message to the current transmitter requesting the channel. The interrupt itself is little more than a CC2420 message header, the *type* field denotes it to be an interrupt and the *src* field identifies its origin. On receipt of such an interrupt, the receiver immediately sends an interrupt acknowledgment and enters a back-off period. The acknowledgment informs the node that the channel has been conceded and its high priority transmission commences.

In the absence of an acknowledgment the sender concludes that the interrupt or its acknowledgment has been lost and enters a back-off phase before trying again. Since there is still a high priority message that needs to be sent the length of this back-off is selected to be shorter than the standard congestion back-off.

VI. APPLICATION AREAS

There are a number of application scenarios that can benefit from the outlined priority interrupt mechanism. In particular, the priority interrupt mechanism can be used to (1) ensure fast delivery of important application messages, (2) implement fairness regarding channel access.

We are currently using TinyOS with the FrameComm MAC layer in a number of different WSN projects. Within these projects the outlined priority interrupt feature is helpful in supporting the application scenarios efficiently.

A. Physical Intrusion Detection (Prioritization)

We are experimenting with a physical intrusion detection system used to secure an office building [17]. A number of wireless intrusion detection sensors are distributed in the building and report their observations to a central sink for data analysis. In addition, each node sends periodic heart-beat messages to indicate correct operation. The sink generates an alarm if an intrusion or system failure is detected and security personnel are informed to deal with the incident.

To ensure timely detection of intruders it is necessary to transport messages with positive detection events with the highest priority. Heart-beat messages and network maintenance messages containing routing or key update material are not time critical. Thus, an efficient network-wide scheduling mechanism is needed.

Part of the solution is a priority scheduler on each node which is able to correctly schedule messages locally. However, a scheduler alone is not sufficient as it is not possible to claim the channel immediately if another node is already in the process of transmitting a low priority message. For example, we frequently observe the case where a key update message is sent from the sink to all nodes (broadcast, downstream), blocking the channel needed for an event detection to be sent to the sink (unicast, upstream). Priority interrupts in FrameCommHPI allow us in such cases to clear the path for an important message traveling upstream.

B. Car Park Management (Fairness)

We constructed and evaluated a wireless sensor network used to monitor the usage of car park spaces [18]. Sensor nodes are deployed on the ground and employ a magnetic field detector to decide if a car is parked in this spot. Subsequently, detection messages are routed towards the sink.

Our study showed that link quality fluctuates heavily and that for extended periods communication between adjunct nodes can be disrupted. Thus, a node might have to try several times to transmit a message successfully to the next hop. In addition, a node might also have to back off as another node gained access to the channel first. Thus, messages routed through particular nodes can experience a very high delay.

Priority interrupts in FrameCommHPI can help to deal with such situations. The priority of a message can be set dynamically, representing the time a node has spent trying to transmit a particular message. Thus, the message priority increases with successive unsuccessful transmissions or back-offs. As a result, nodes which have to deal with bad channel conditions will obtain higher priority in accessing the medium.

VII. EXPERIMENTAL EVALUATION

The primary goal of the experiments is to quantify the reduction in channel access time for high priority traffic as a result of using FrameCommHPI.

A. Evaluation Metrics

The main measurement of interest is the node-to-node delivery latency, t , of messages. However, without a common time

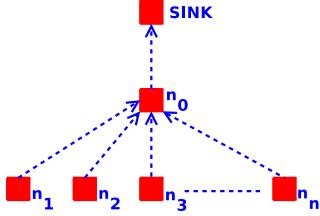


Fig. 5. Experiment Setup.

source or complex synchronization mechanism it is difficult to measure this latency. With this in mind it was decided to measure the latency locally on each node by comparing the time the node decides to schedule a message for transmission, t_1 , and the time the node receives an acknowledgment, t_2 , ($t = t_2 - t_1$). More specifically, within the TinyOS implementation we measure the period between the application calling a *send* and receiving a *sendDone* signal. While there is some overhead associated with a message being sent down the TinyOS communication stack and the *sendDone* being delivered, this time does not vary between experiments or across nodes. The message size is constant throughout and the nodes are not carrying out any other operations or activities that could affect these times. As a result the only variable is the time taken to access the channel and deliver the message. The number of packet losses, r , (given in percent) is also of interest and is determined.

B. Experiment Setup

The experiments were carried out using Moteiv Tmote Sky and Tmote Invent sensor nodes running TinyOS 2.0.2. A small tree topology consisting of some leaf nodes, a forwarding node and base station (sink) as shown in Fig. 5 was used. The leaf nodes and the forwarding node were duty cycled while the base station was always on.

Each experiment begins by sending a control message requesting a number of messages from each leaf node (n_1 to n_n). The leaf nodes then send a series of messages to the forwarding node n_0 that sends them to the sink where they are uploaded to a PC for analysis. The payload of each message consists of a node ID, a sequence number and the measured message send time of the previous message. The first 100 messages from each node are taken and from these the average latency t and loss rate r are calculated. The measurement is repeated 5 times for each measurement point. A 2% Duty Cycle was employed giving a Duty Cycle Period, $P = 600ms$, for FrameComm and $P = 550ms$ for LPL.

FrameCommHPI's high priority interrupt feature only comes into effect when there is contention on the channel. Thus, a message generation interval of $n \cdot 500ms$ was chosen, where n is the number of leaf nodes. A fixed generation interval would introduce a bias towards one node as it would generate its message first each time, to prevent this the interval included randomization. The desired average generation interval, txr , is included in the control message. Each

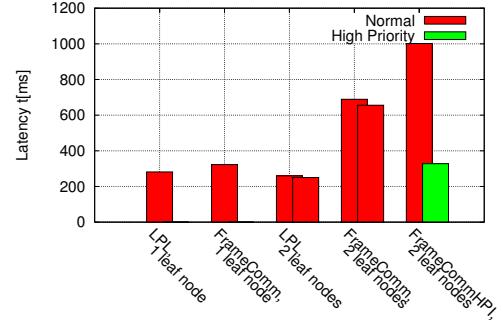


Fig. 6. Experiment 1, Latency t .

individual interval, txr_i , is calculated with the formula $txr_i = (rand \% txr) + \frac{txr}{2}$. This gives a value between $\frac{txr}{2}$ and $\frac{3txr}{2}$, averaging out to txr over the course of the experiment.

C. Experiment 1

Initially the experiment was run with a single leaf node n_1 generating messages to get a base latency t_b , that is a latency t without channel access delays introduced by contention. Subsequently, the experiment was repeated with two leaf nodes generating messages for FrameComm, LPL and FrameCommHPI. Messages are generated at a random interval that average at $2 \cdot 500ms = 1000ms$. The results are shown in Table I and Fig. 6, the large values for standard deviation is due to the scheme being Duty Cycled.

Single Leaf Node: With a single sender LPL slightly outperforms FrameComm in terms of latency, t . This was expected as LPL has a slightly smaller duty cycle period than FrameComm. The measured average latency $t_{b,FrameComm} = 322.81ms$ and $t_{b,LPL} = 281.65ms$ were slightly greater than the expected $\frac{P}{2}$, confirming t as valid metric. With a single leaf node, no losses were recorded for either protocol.

Two Leaf Nodes: The introduction of a second leaf node generating messages illustrates the effect of the differences between LPL and FrameComm and especially FrameCommHPI.

LPL's message send times are largely unchanged but there is a significant increase in packet losses. FrameComm's latency increases significantly but the packet losses remain low. This result shows that the current TinyOS LPL implementation suffers considerable losses where contention is present. The increased latency with FrameComm can be attributed to nodes trying to access a busy channel.

When FrameCommHPI is introduced it is clearly evident that the delay for Node 2, the node generating high priority messages, is significantly reduced. The average values are only marginally above the target $t_{b,FrameComm}$ set by the single sender node using FrameComm and is very close to $\frac{P}{2}$ as expected. This additional latency can be attributed to the necessary exchange of interrupts and interrupt acknowledgment.

LPL v FrameComm: Fig. 6 shows that latency in LPL is relatively unaffected by an adding congestion when compared to FrameComm. This would seem to be in LPL's favor but

	Latency t [ms]				Loss Rate r [%]			
	Node1		Node2		Node1		Node2	
	Average	Std Dev	Average	Std Dev	Average	Std Dev	Average	Std Dev
LPL _(1 sender)	281.65	159.52	-	-	0.0	0	-	-
FrameComm _(1 sender)	322.81	166.70	-	-	0.0	0	-	-
LPL _(2 senders)	259.50	173.14	250.02	167.87	26.6	6.69	21.0	4.00
FrameComm _(2 senders)	688.98	332.49	655.80	315.73	0.6	0.55	0.4	0.55
FrameCommHPI _(2 senders)	1001.93	658.39	327.85	192.58	0.4	0.55	0.4	0.55

TABLE I
EXPERIMENT 1, LATENCY t , LOSS RATE r .

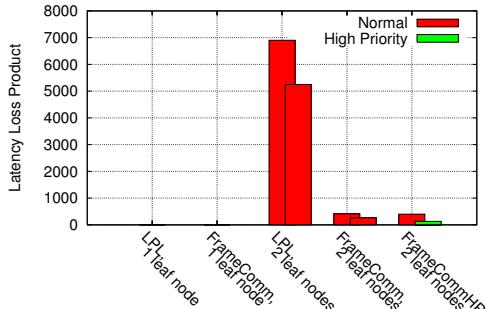


Fig. 7. Experiment 1, Latency Loss Product.

this is not a complete picture as it does not factor in losses. Fig. 7 on the other hand reflects both latency and losses and it clearly shows LPL has problems when multiple senders are present.

While LPL is energy efficient and reliable with a single sender it suffers considerable losses when the network is congested. For some applications this may not be an issue but for those with high priority messages or critical data such losses would be a concern. Event based systems are most likely to suffer since it is probable that any nodes in the vicinity of a high priority event will detect it and attempt to report it, resulting in contention.

FrameComm continues to be very reliable in the presence of heavy congestion due to some key differences between its design and that of LPL. This is mainly as a result of LPL's CCA mechanism and its backoff as described in Section V-A.

D. Experiment 2

Experiment 2 examines the behavior of FrameCommHPI in terms of average latency t with various combinations of normal and high priority senders. This gives an indication of how FrameCommHPI performs in a larger scale multi-hop network with a mixture of priorities, particularly in upstream forwarding nodes where data converges approaching the sink. Each run of this experiment had four leaf nodes generating messages. Messages are generated at a random interval that average at $4 \cdot 500ms = 2000ms$. To begin with all four leaf nodes sent messages of normal priority. With each subsequent variant the priority of one sender was increased, giving three normal and one high priority, then two normal, two high

and finally one normal and three high. The results of this experiment are shown in Fig. 8. A cumulative distribution for the latency of both normal and high priority traffic is presented for each variant of the experiment.

With a single high priority sender an average latency of very close to $\frac{P}{2}$ is achieved, despite the extra network traffic. This value increases with each additional high priority sender, however even with three high priority senders the average latency for high priority messages is still half that of when all messages priorities are equal. A more dramatic jump for the latency of low priority messages can also be seen. This is expected as a larger proportion of high priority messages leads to more interruptions.

VIII. CONCLUSION & FUTURE WORK

In this paper a concept for prioritizing messages using interrupts in Duty Cycle communication schemes is presented. An implementation called FrameCommHPI is described and evaluated along with practical application areas that would benefit from its use. As shown, the channel access times for critical data can be effectively reduced and message transport latency is improved. In the future, we plan to combine the presented interrupt mechanism with an adaptive duty cycle scheme to further improve data delivery latency.

Acknowledgments: Mr. O'Donovan is supported by Microsoft Research through its European PhD Scholarship Programme and the EMBARK Initiative of the Irish Research Council for Science, Engineering and Technology.

REFERENCES

- [1] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. Proceedings of the 2nd ACM Conference on Embedded Networked Sensor Systems, 2004.
- [2] J. Benson, T. O'Donovan, U. Roedig, and C. Sreenan. Opportunistic Aggregation over Duty Cycled Communications in Wireless Sensor Networks. Proceedings Of ACM/IEEE International Symposium on Information Processing in Sensor Networks (IPSN), 2008.
- [3] Tinyos 2.0.2. <http://www.tinyos.net>.
- [4] Motieiv Tmote Sky & Tmote Invent motes. <http://www.moteiv.com>.
- [5] M. Buettner, G. V. Yee, E. Anderson and R. Han. X-MAC: A Short Preamble MAC Protocol for Duty-Cycled Wireless Sensor Networks. Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys), 2006.
- [6] R. Musaloiu-E, C. M. Liang, A. and Terzis. Koala: Ultra-Low Power Data Retrieval in Wireless Sensor Networks. Proceedings Of ACM/IEEE International Symposium on Information Processing in Sensor Networks (IPSN), 2008.

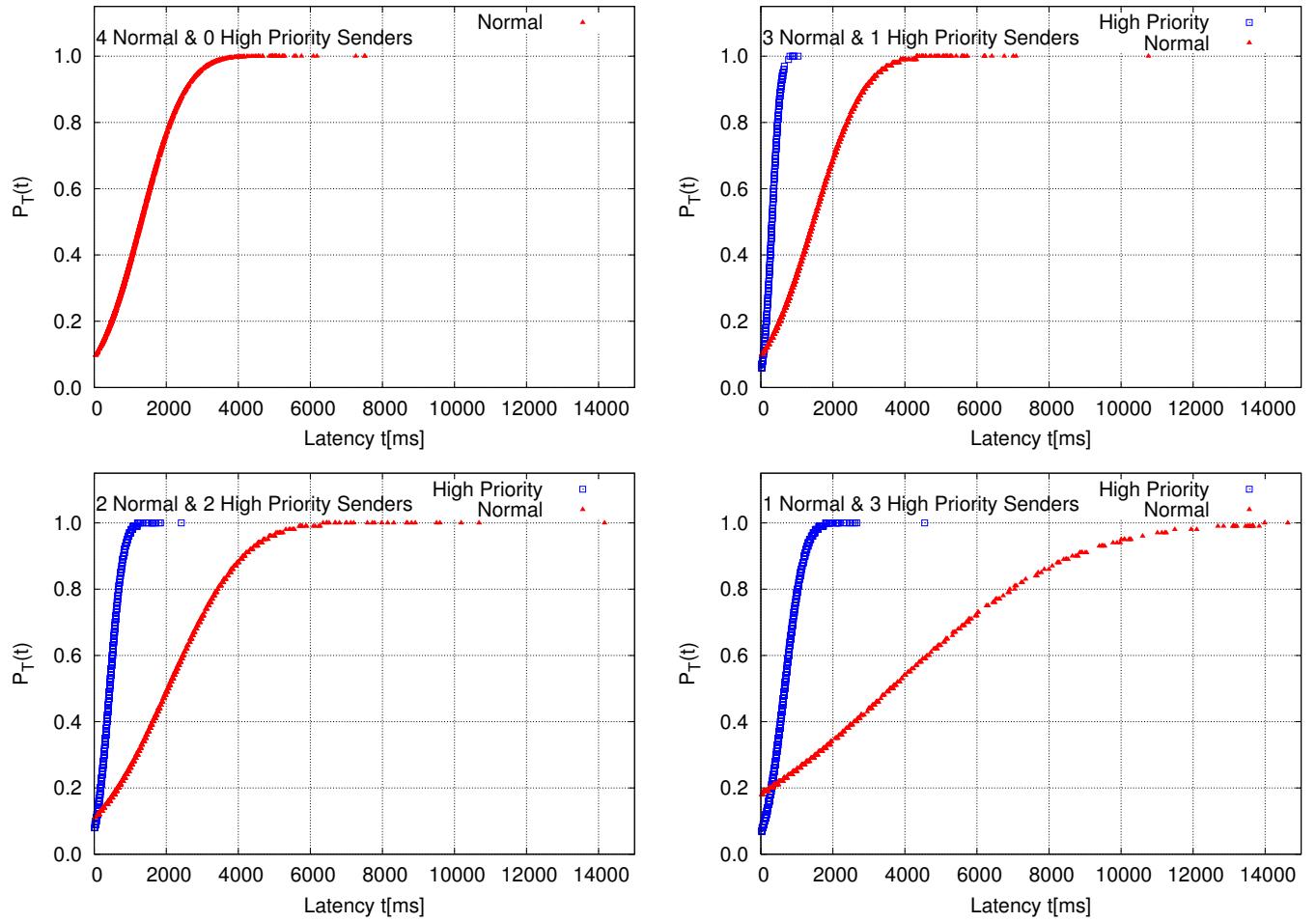


Fig. 8. Experiment 2, Cumulative Distribution, $P_T(t)$ v Latency t [ms].

- [7] K. K. Chintalapudi and L. Venkatraman. On the Design of MAC Protocols for Low-Latency Hard Real-Time Discrete Control Applications over 802.15.4 Hardware. Proceedings Of ACM/IEEE International Symposium on Information Processing in Sensor Networks (IPSN), 2008.
- [8] P. Ansel, Q. Ni, and T. Turletti. An Efficient Scheduling Scheme for IEEE 802.11e. Proceedings of IEEE Workshop on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks, 2004.
- [9] A. Firoze, L. Jun and L. Kwong. PR-MAC A Priority Reservation MAC Protocol For Wireless Sensor Networks. Proceedings of International Conference on Electrical Engineering (ICEE), 2007.
- [10] Y. Xiao. Enhanced DCF of IEEE 802.11e to support QoS. Proceedings of IEEE Wireless Communications and Networking (WCNC), 2003.
- [11] J. W. Fonda, M. Zawodniok, S. Jagannathan and S. E. Watkins. Adaptive Distributed Fair Scheduling and Its Implementation in Wireless Sensor Networks Systems. IEEE International Conference on Man and Cybernetics (SMC), 2006.
- [12] O. Chipara, C. Lu, C. and Roman. Real-Time Query Scheduling for Wireless Sensor Networks. Proceedings of the 28th IEEE International Real-Time Systems Symposium (RTSS), 2007.
- [13] K. Karenos, V. Kalogeraki and S. V. Krishnamurthy. A Rate Control Framework for Supporting Multiple Classes of Traffic in Sensor Networks. Proceedings of the 26th IEEE International Real-Time Systems Symposium (RTSS), 2005.
- [14] A. Krohn, M. Beigl, C. Decker and T. Zimmer. TOMAC - Real-Time Message Ordering in Wireless Sensor Networks Using the MAC Layer. Proceedings of the 2nd Intern'l Workshop on Networked Sensing Systems, 2005.
- [15] J. Brown, J. Finney, C. Efstratiou, B. Green, N. Davies, M. Lowton and G. Kortuem. Network interrupts: supporting delay sensitive applications in low power wireless control networks. Proceedings of the 2nd Workshop on Challenged Networks, 2007.
- [16] A. Barroso, U. Roedig, and C. J. Sreenan. Use of Framelets for Efficient Transmitter-Receiver Rendezvous in Wireless Sensor Networks. Proceedings of the 5th International IEEE Workshop on Wireless Local Networks, 2005.
- [17] A. Chung and U. Roedig. DHB-KEY - A Diffie-Hellman Key Distribution Protocol for Wireless Sensor Networks. Adjunct Proceedings (Posters) of the 5th IEEE European Workshop on Wireless Sensor Networks (EWSN), 2008.
- [18] J. Benson, T. O'Donovan, U. Roedig, P. O'Sullivan, C. Sreenan, J. Barton, A. Murphy, and B. O'Flynn. Car-Park Management using Wireless Sensor Networks. Proceedings of the First IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SENSEAPP), 2006.